CSC 574
Computer and Network Security

Firewalls and IDS

Alexandros Kapravelos
kapravelos@ncsu.edu

(Derived from slides by William Robertson)

SpiderLabs® Blog

Neutrino Exploit Kit Not Responding – Bug or Feature?

February 4, 2016 | Posted By Daniel Chechik | Comments (0)

Share:

A couple of weeks ago we were looking at some exploit kits in one of our lab environments and noticed a decline in the number of Neutrino instances we're seeing. This sent us on yet another journey to investigate Neutrino and understand some weird behaviours we were observing. This blog post will tell the story of Neutrino, confused researchers, and OS fingerprinting.

A researcher's first thought when an exploit kit suddenly behaves strange is that the exploit kit developer made some changes or added some features to reduce the exploit kit's

Firewalls

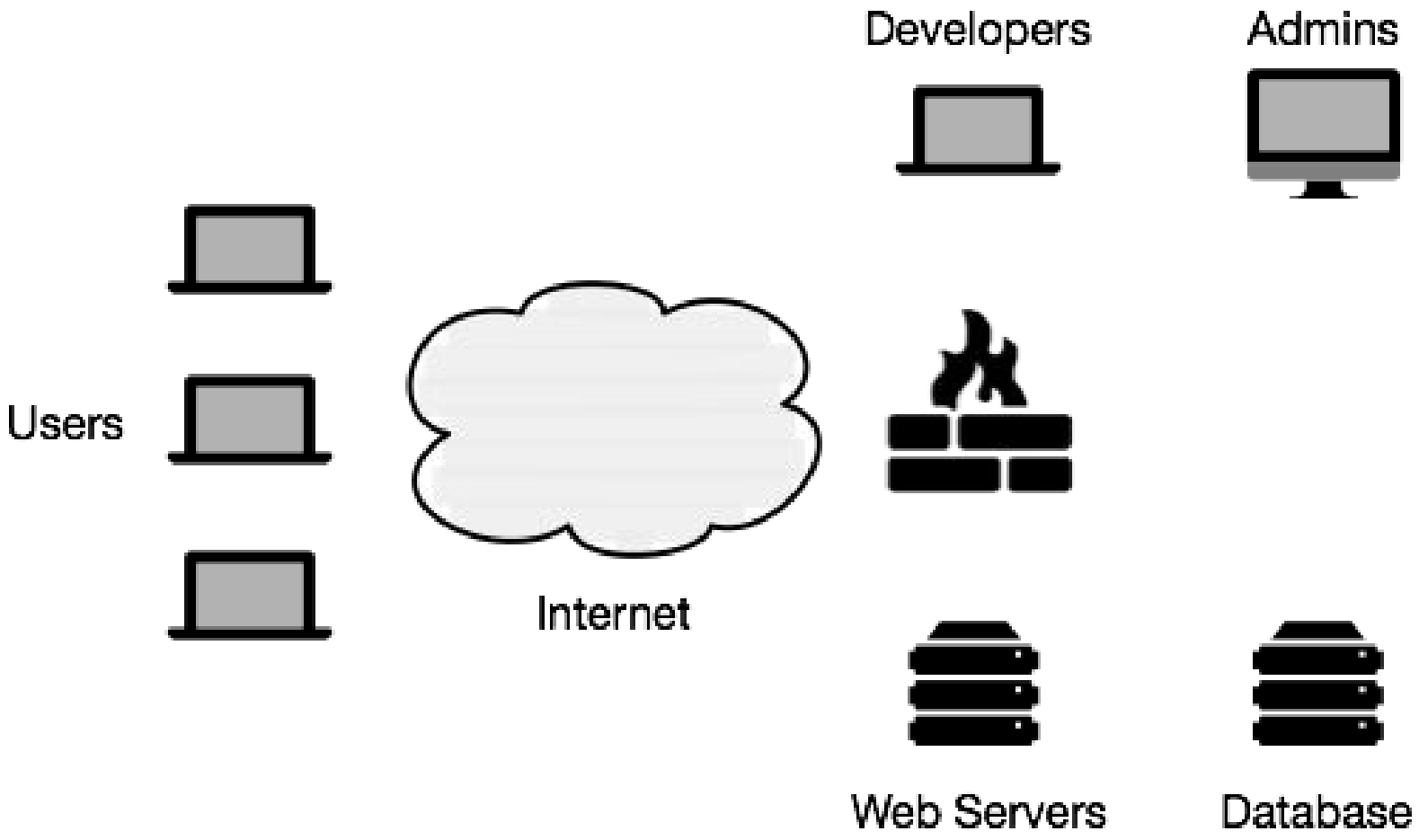
- Recognized early on that network-level access control would be useful
  - e.g., ensuring internal services remain internal, defending known-vulnerable machines
- Firewalls inspect network traffic and filter/modify it according to some predicates (ruleset)
- Several classifications
  - Packet filtering, stateful filtering, application-layer filtering

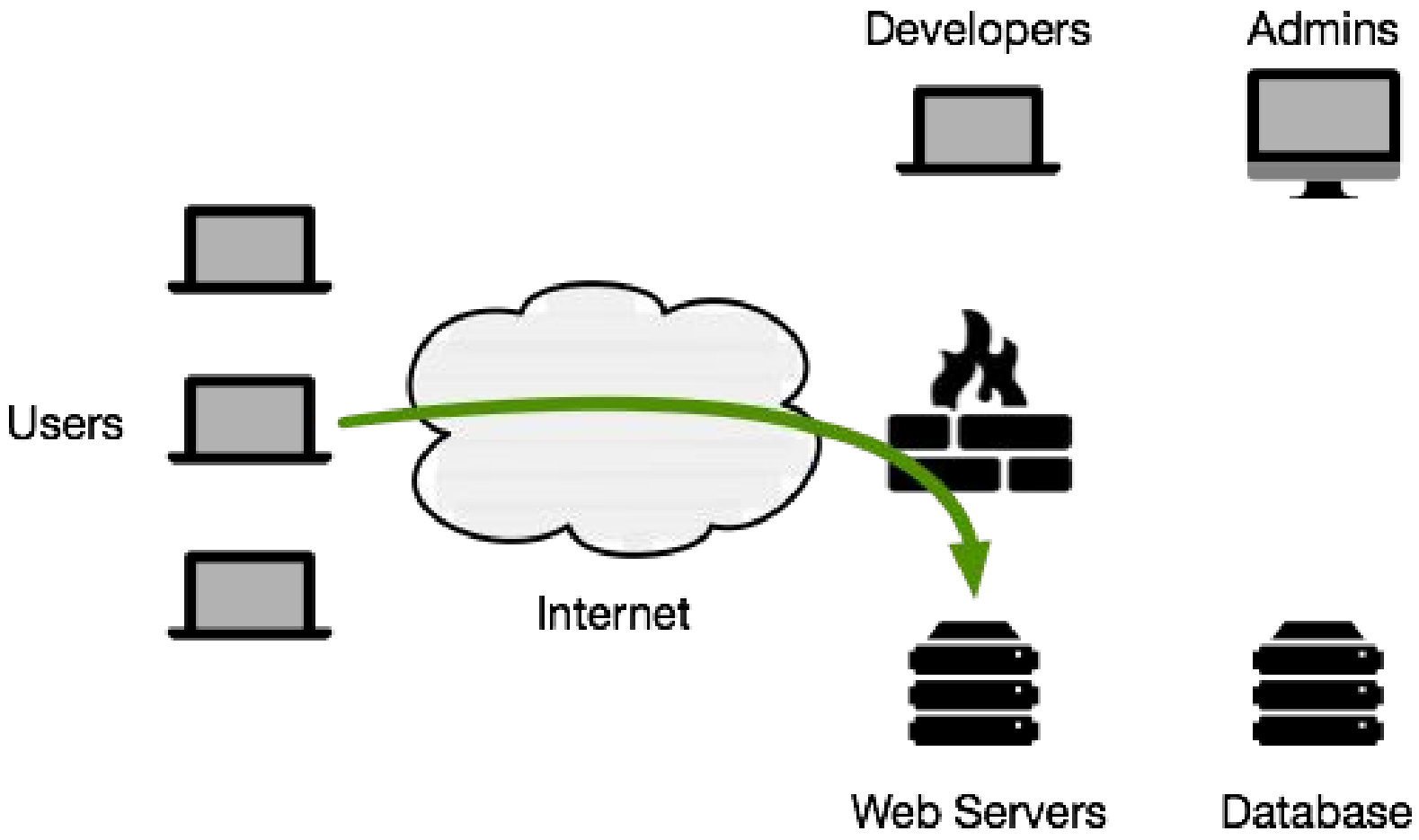
Packet Filtering

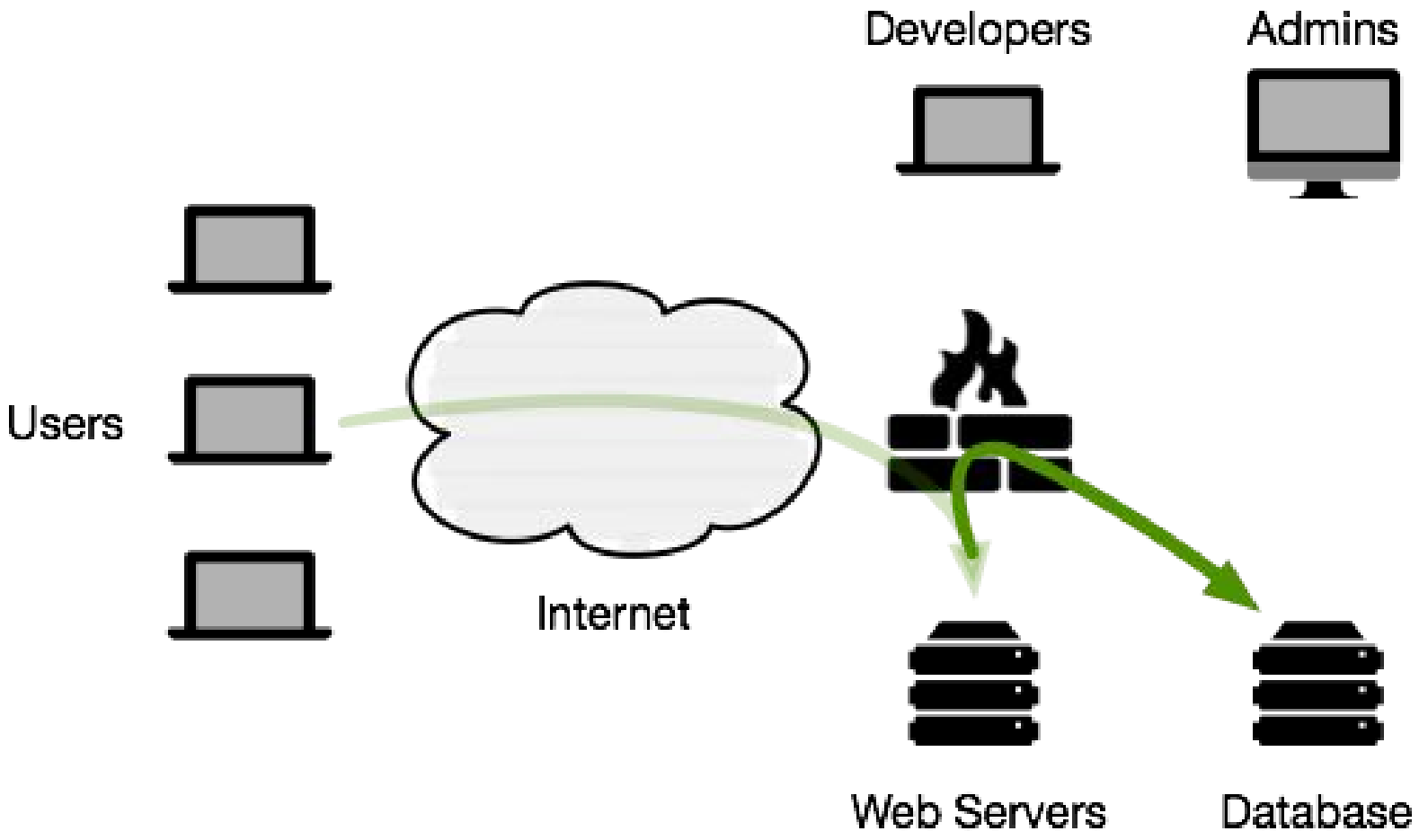
- Application of filtering predicates over individual packets
  - IP addresses
  - IP protocol
  - Packet sizes
  - TTLs
  - Ports
- Straightforward to implement
  - But, not capable of detecting some attacks

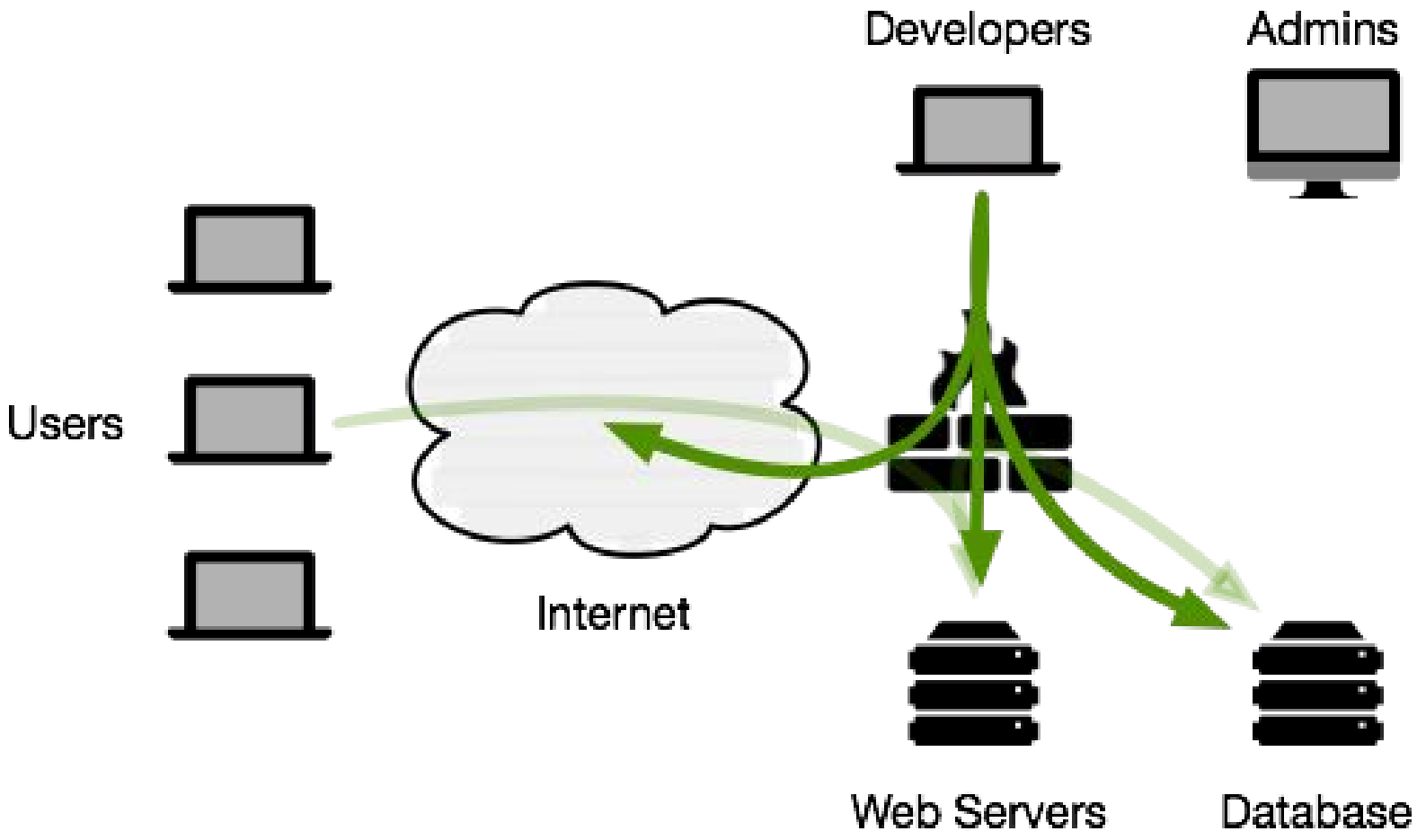
Stateful Filtering

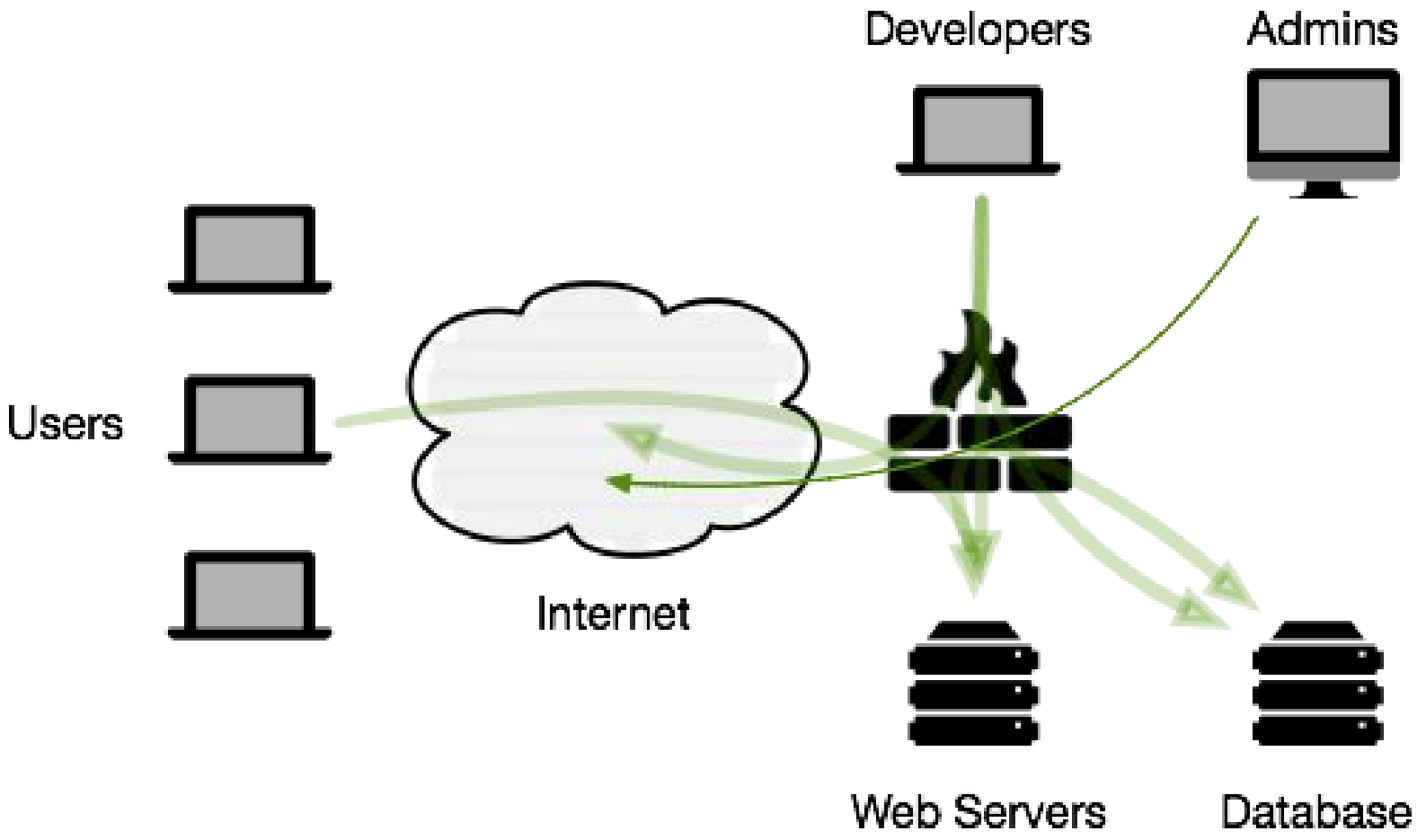
- Firewall maintains state across packet sequences
- Application of filtering predicates over network streams
  - Connection state (INVALID, NEW, ESTABLISHED, RELATED)
  - Temporal information
  - Tagging
- More powerful detection primitives
- Drawbacks
  - Prone to availability attacks

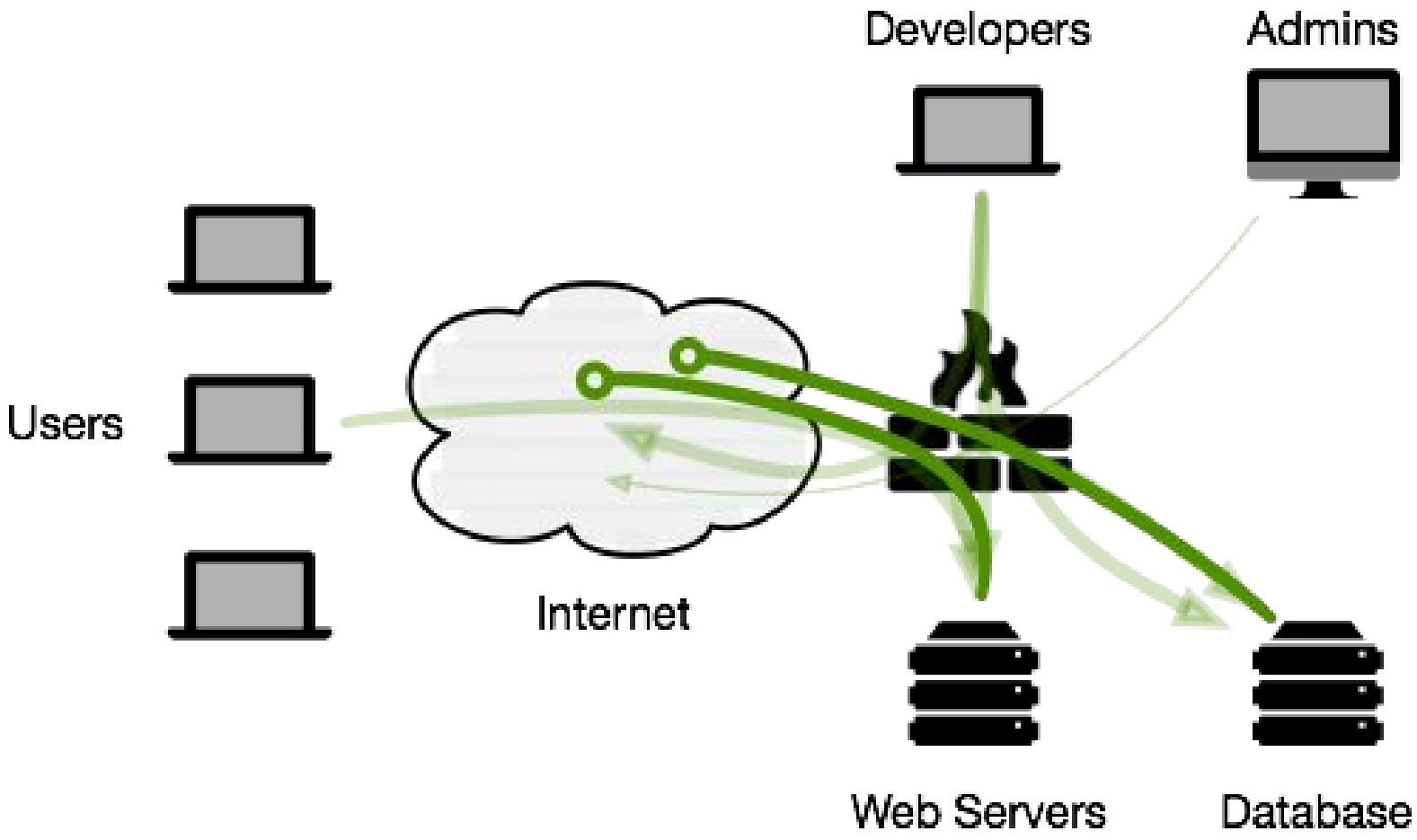

Developers
Admins
Users
Internet
Web Servers
Database

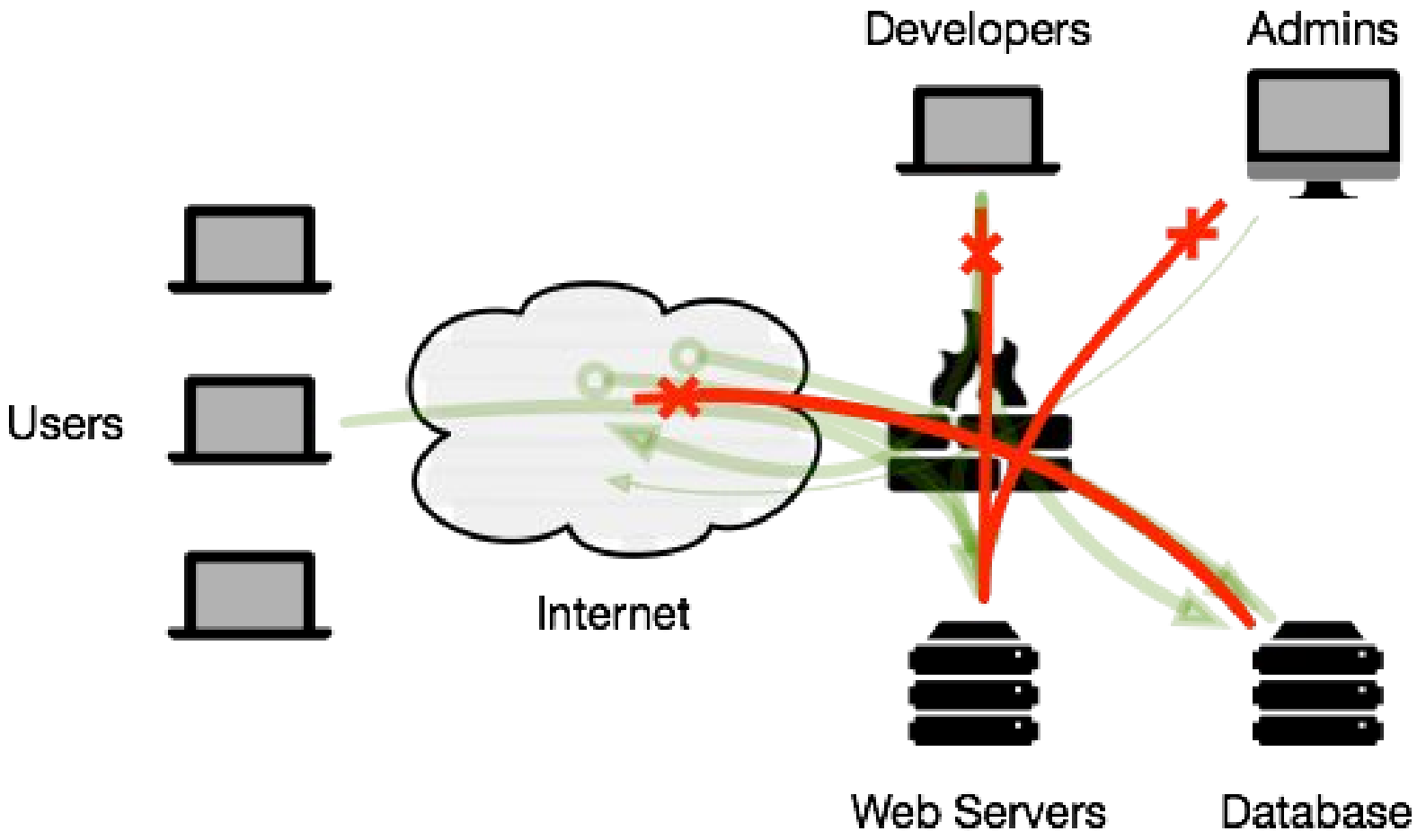

Developers
Admins
Users
Internet
Web Servers
Database


Developers
Admins
Users
Internet
Web Servers
Database


Developers
Admins
Users
Internet
Web Servers
Database


Developers
Admins
Users
Internet
Web Servers
Database


Developers
Admins
Users
Internet
Web Servers
Database


Developers
Admins
Users
Internet
Web Servers
Database

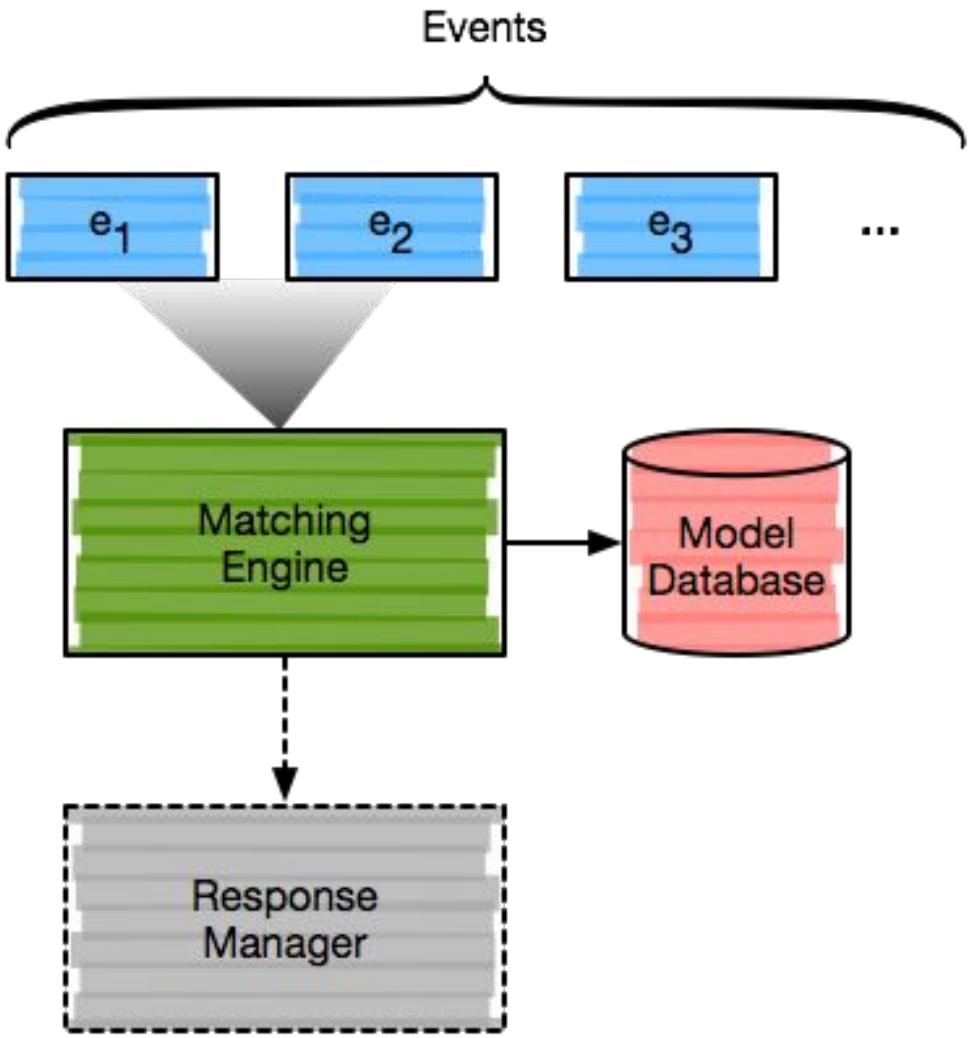
Firewall Attacks

- Firewalking
  - Often useful to understand firewall policies
  - Traceroute-like technique
    - i.e., increment TTL until firewall discovered, increment once more, check for ICMP errors
- Source port scanning
- Desynchronization (similar to IDS)

Intrusion Detection

- General term for detecting attacks against systems
- Embodiment of *detection* approach to security
  - Assume that attacks will occur, develop techniques to identify and counter threats
- Many ways to characterize intrusion detection systems (IDS)
  - Domain (e.g., network, host, application-based)
  - Misuse- vs. anomaly-based
  - Stateless vs. stateful

IDS Components



Detection Theory

	Event Occurred	Event Did Not Occur
Event Detected	True positive	False positive
Event Not Detected	False negative	True negative

- IDSs are essentially binary classifiers
- Detection theory considers two boolean variables
  - Whether an event occurred
  - Whether an event was detected

Events (Detection Domains)

- Detection is performed over some abstract event stream
- Goal is find evidence of malice over sequences of events
  - Network packet headers, network packet contents, network streams
  - System calls, function calls, arbitrary control-flow transfers, full execution traces
  - Application-level events (e.g., HTTP messages, user logins)

Misuse Detection

```
alert tcp any any -> 192.168.0.0/24 80 \
   (content: |90 90 90 90|"; msg: "Shellcode!";)
```

- Misuse detection: Search for direct evidence of attacks
  - Essentially the "blacklist" approach
- Models of malice encoded as *signatures*
- Typical advantage: Capable of precise matching
- But, prone to *false negatives*

Anomaly Detection

```
benign any any -> 192.168.0.0/24 80 \
    (content: ascii; violation-msg: "Non-ASCII!";)
```

- Anomaly detection: Identify "previously unseen" behavior
  - Presumption is that unknown behavior is indicative of malice
  - Essentially the "whitelist" approach
  - Often combined with machine learning of models
- Typical advantage: Ability to detect 0-day attacks
- But, prone to *false positives*

Adversarial Machine Learning

- Assumption: Training data is synthetic, *or* real data but only contains good behavior
  - What if training set contains attacks, or not *all* benign behavior?
- Assumption: Notion of "normal" remains constant, *or* models can be updated as conception of normality changes
  - What if "normal" isn't constant?
  - What if attacker can influence model updating?
- Assumption: Attacks are distinguishable from normal behavior using models
  - What if an attacker can create "normal"-looking attacks?

Stateless vs. Stateful

- Early systems were stateless
  - Each event is considered in isolation
  - Stateless techniques are efficient, but prone to evasion
- More advanced techniques are stateful
  - Allows more precise, complicated matching predicates
  - But, can lead to denial-of-service against IDS
  - Stateful modeling still difficult to get right

Limitations

```
def malware_contradiction():
    if not D(malware_contradiction):
        launch_attack()
```

- IDS is known to be a difficult and generally intractable problem
  - Fred Cohen, reduction to halting problem
- In practice, false positives are the limiting factor for an IDS, not false negatives!
  - Base rate of attacks is low in most environments
  - Even minuscule false positive rates are magnified

Base Rate Fallacy

- Let A, I be two boolean random variables
  - I – an event represents an intrusion (w.l.o.g.)
  - A – an alert is raised
- $P(A|I)$ – probability an alert is raised if an intrusion occurs
- $P(A|\neg I)$ – false positive
- $P(I|A)$ – if there is an intrusion, was there an alert?

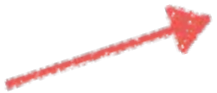
Base Rate Fallacy

$$P(I) = 2 \times 10^{-5}$$

$$P(A|I) = \frac{P(A) \cdot P(I|A)}{P(I)}$$

$$P(I|A) = \frac{P(I) \cdot P(A|I)}{P(I) \cdot P(A|I) + P(\neg I) \cdot P(A|\neg I)}$$

$$= \frac{2 \times 10^{-5} \cdot P(A|I)}{2 \times 10^{-5} \cdot P(A|I) + 0.99998 \cdot P(A|\neg I)}$$



Detection rate dominated by false positive rate! (If you want a high detection rate, most of your alerts will be false positives)

Base Rate Fallacy Example

- Consider the case of a medical test that has 99% ac
  - When 100 people have the condition, 99 decisions positive
  - When 0 people have the disease, 99 decisions are negative
- During consultation, the doctor tells you he has good news and bad news
  - Bad news: You tested positive
  - Good news: Only 1 in 10,000 have the condition
- *What is the probability you have the condition?*

P(A|I) = 0.99

P(¬A|¬I) = 0.99

P(I) = 0.0001

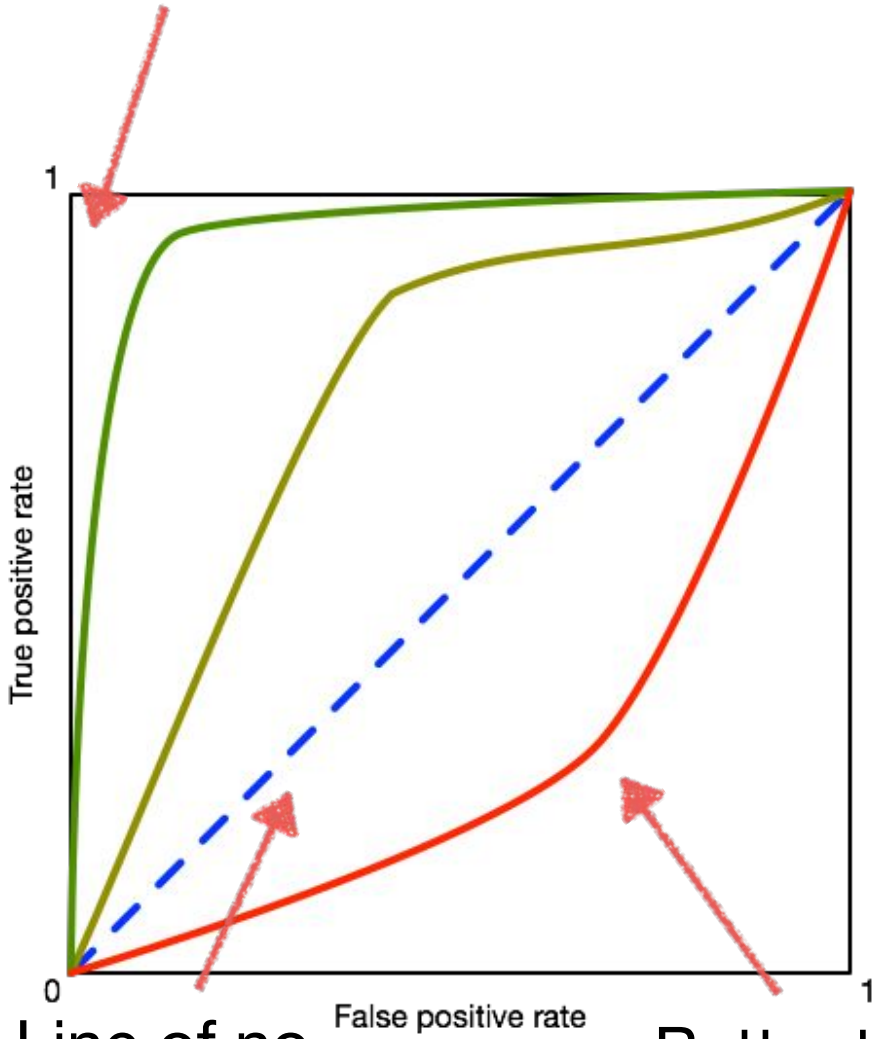
Base Rate Fallacy Example

$$
\begin{aligned}
P(I|A) &= \frac{P(I) \cdot P(A|I)}{P(I) \cdot P(A|I) + P(\neg I) \cdot P(A|\neg I)} \\
&= \frac{0.0001 \cdot 0.99}{0.0001 \cdot 0.99 + (1 - 0.0001) \cdot 0.01} \\
&= 0.0098 \\
&\approx 1\%
\end{aligned}
$$

False Positives

- A non-trivial false positive rate has several negative effects
  - Obscures true attacks
  - Induces user fatigue
  - Can have a high associated cost
- Positive correlation between IDS visibility into monitored system and reduction of false positives
  - Less visibility leads to potential for desynchronization and evasion

IDS Evaluation



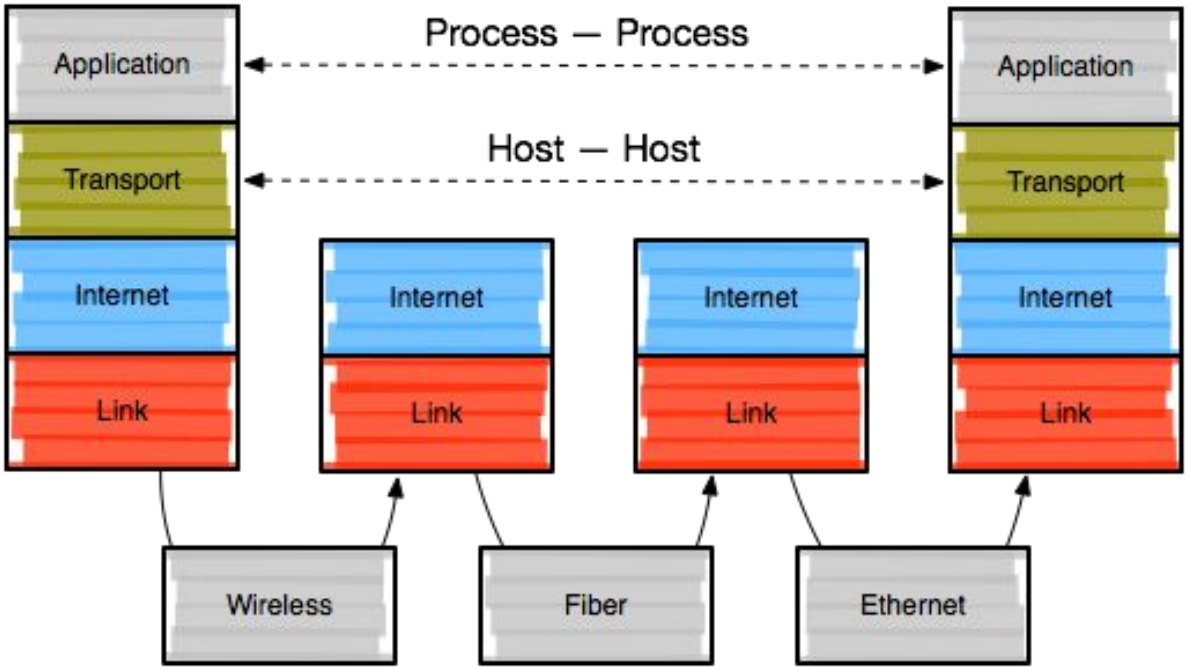
- ROC plots commonly used to evaluate or compare IDSs
- IDS is run on test data
- Repeated for various detection thresholds
- Plot TPs against FPs
- Indicates what TPR can be expected for a given FPR

Network-based IDS

Network Intrusion Detection

- Goal: Detect attacks on the wire
- Match detection models against network traffic
- Approach is desirable because it (potentially) protects many machines
- But, there is an associated difficulty in accurately modeling state on those machines...

Network IDS



- IDS can be applied to each layer of the network stack

Example: Unique IP Addresses

- Let's consider one point in the design space: detecting anomalous network behavior
  - In particular, how many unique IP addresses a user contacts in a day
  - We want to learn a model from training data that captures the *normal* behavior of a user (i.e., in a day, user *x* contacts around *y* unique IP addresses)
  - How might we construct a model and how would we apply it?

Example: Unique IP Addresses

$$P(|X - \mu| > t) < \frac{\sigma^2}{t^2}$$

- One very simple approach: apply Chebyshev's inequality
  - Non-parametric upper bound on probability that the difference between a random variable *X* and a learned μ exceeds a threshold *t*
  - Since we only care about increases in number of addresses as an attack, we treat the inequality as one-sided

Example: Unique IP Addresses

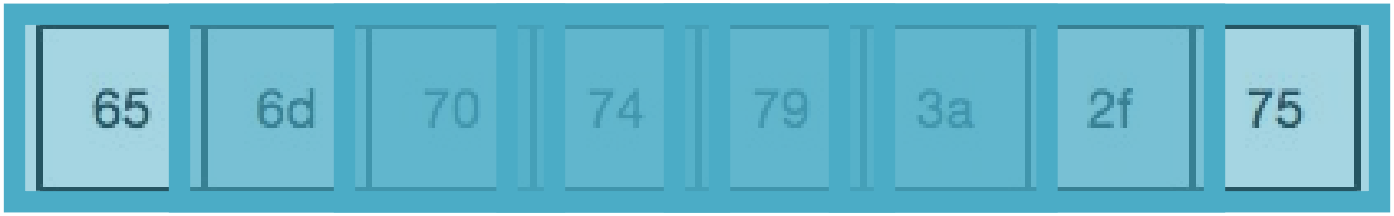
$$P(|X - \mu| > t) < \frac{\sigma^2}{t^2}$$

- Why are some advantages of this approach?
  - Can be computed from packet headers
  - Easy to acquire model parameters and evaluate model on new observations
  - Non-parametric (no assumption on underlying distribution)
  - Loose upper-bound (lower chance of false positives)

Anagram

- What if we want to detect more complicated attacks in a general way?
  - Idea: anomaly detection over network *content*
- Anagram uses a combination of *n*-grams and Bloom filters to efficiently identify unknown, possibly malicious, network traffic
  - *n*-grams constructed by sliding window of size *n* over event sequences (packet content)
  - Semi-supervised: Extracted *n*-grams for both positive and negative examples stored in Bloom filters

n-grams



83	3d	98	fa	08	00	00	75

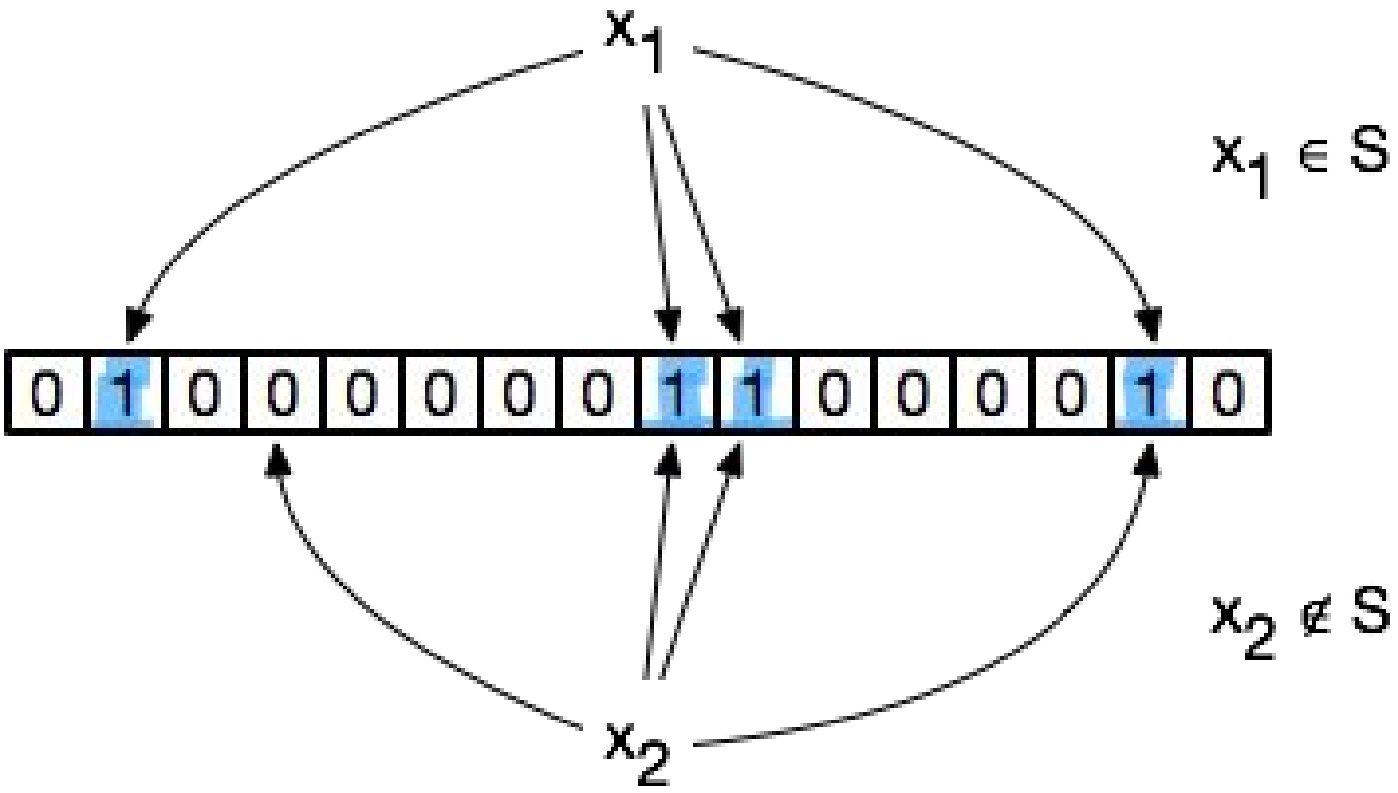
n-gram Models

- Recording a frequency distribution is memory-intensive!
  - 256^n possible *n*-grams (size of feature space)
  - Might want multiple models for different protocols
  - Need a space-efficient way to record model
- Can we even estimate the true empirical distribution accurately in an efficient way?
  - Anagram performs per-packet classification, but the feature vector drawn from an individual packet is very sparse

Bloom Filters

- Instead of a frequency distribution, Anagram uses Bloom filters
- Bloom filters are a *probabilistic* data structure for recording value sets
  - Bit array of m bits, k hash functions $\{f_1,...,f_k\}$
- Insertion and membership implemented by computing $f_i(x)$ for all f_i and setting / testing corresponding bit set
  - Membership: If all bits set, element *might be* present, otherwise element is definitely not present
- Careful selection of m, k, f_i required to achieve desired FPR and avoid saturation

Bloom Filters



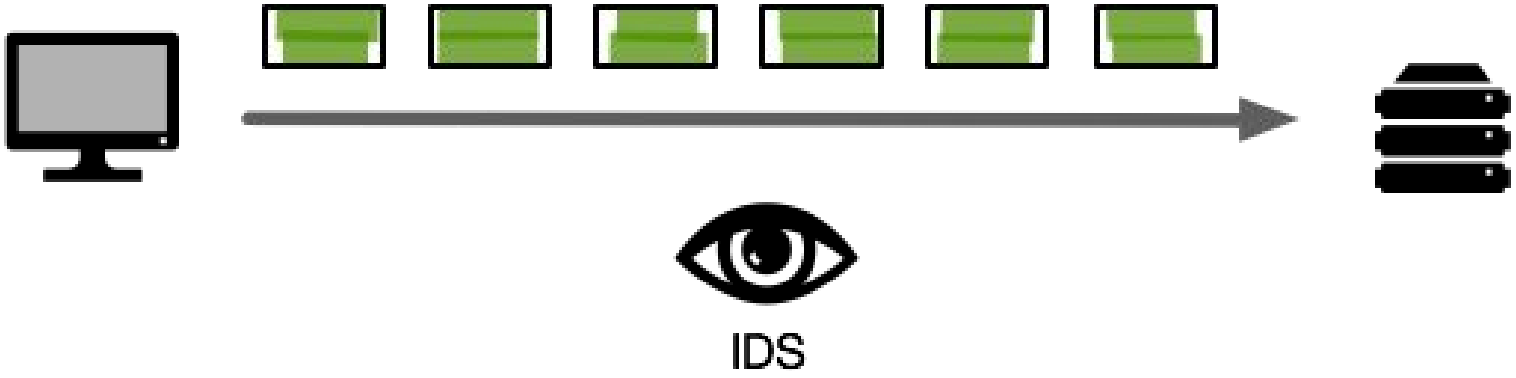
Anagram

- Bloom filters used per-packet to compute:
  - N_{new}: number of new *n*-grams
  - T: total number of *n*-grams
- Decision function
  - The score is computed as: $\frac{N_{\text{new}}}{T} \in [0, 1]$
- But, we also need a threshold to discriminate between benign and anomalous scores (how?)
  - Derived through empirical observation

NIDS Evasion

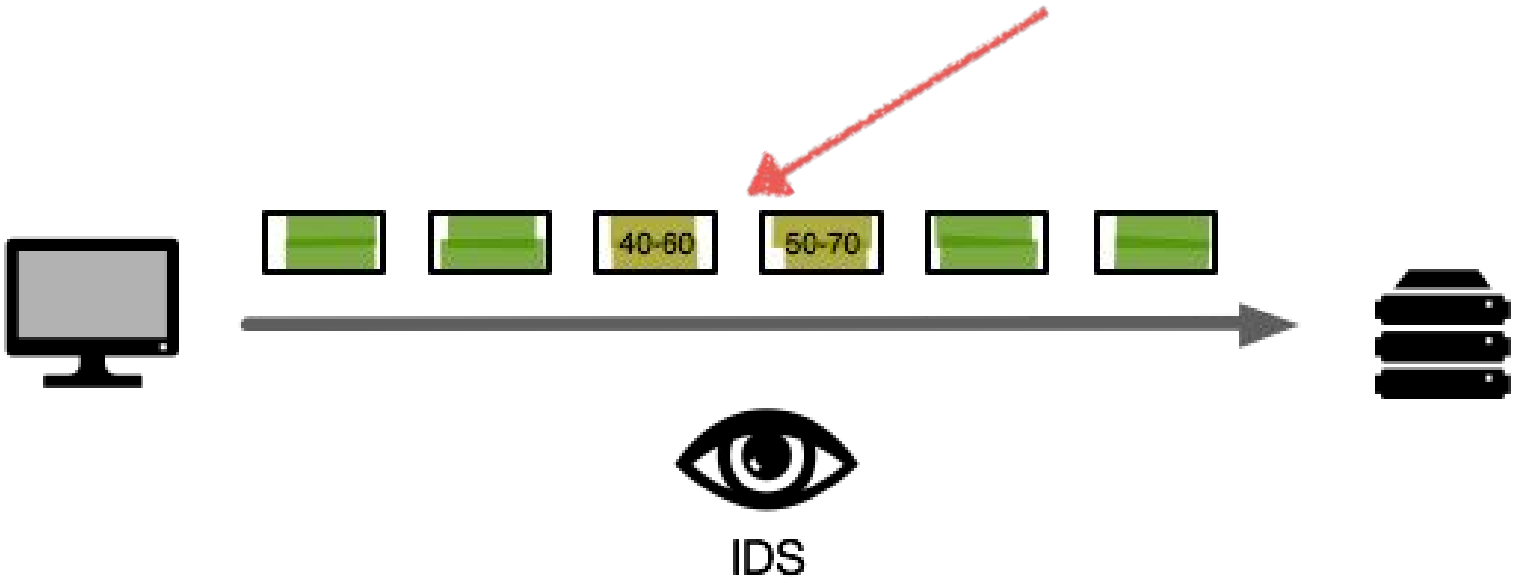
- Many ways to evade network IDS
  - Flooding (actual packets)
  - Flooding (fake attacks, create too many alerts)
- A big problem is desynchronization
  - How does the IDS know the actual state of the monitored hosts and applications?
  - Much imprecision and ambiguity resulting from timers, reassembly, imprecise protocol specification, buggy implementations, ...

NIDS Evasion



NIDS Evasion

Overlapping TCP segments!



Seriously?

October 01, 2014 | Bob Walder

We don't follow up every NSS Labs test with a blog response to a vendor, but after the fun and games following our recent BDS test, we find ourselves in a similar position. This time it is Palo Alto Networks blogging about our NGFW group test, the results of which were published last week and can be found here.

While Lee Klarich's blog was very carefully worded, he never actually addressed the main issue at hand: Palo Alto Networks NGFW misses several critical evasions that leave its customers at risk. The blog did, however, contain some serious inaccuracies that I would like to address:

Palo Alto Networks' claim	NSS' response
"Palo Alto Networks intentionally did not participate in the 2014 NSS Next-Generation Firewall Comparative Analysis report that was recently published. This means that unlike all of the other vendors in the report who configured and tuned their products specifically for this test, there was no input from us on the configuration of our device."	**Participation in an NSS group test is not optional** – if you sell into a particular market, or if our enterprise clients want to see your products tested, then we will test them. It is always worrying when a vendor is so resistant to having its product tested - usually an indication that it's engineers know there is something

Host-based IDS

Host-based IDS

- Host-based intrusion detection integrates detection into the endpoints
  - Has the advantage that evasion becomes much more difficult due to greater insight into monitored system, applications
  - But, is more complex, requires deployment on all systems
- We'll look at a couple of examples

Tripwire

- Tripwire is a simple anomaly-based host IDS
- Intended to defend against literal system intrusions
  - Attacker gains user access to system, elevates to root
  - Replaces system binaries to gain backdoor persistence
- Tripwire monitors changes in system binaries
  - Compute cryptographic hash of key system binaries
  - Later, compare stored hash against computed hash
  - Reference hash values stored on separate, read-only medium

Tripwire

- Tripwire is a simple and (sometimes) effective idea
- Advantages
  - Lightweight, takes advantage of simple invariant to detect many attacks (even 0-days)
- Disadvantages
  - Attack has already occurred!
    - Detection only occurs when hashes are checked
  - Doesn't catch attacks that change system files
  - Requires separate checking system for high assurance

System Call Monitoring

- Let's consider another example: *syscall monitoring*
  - Build an FSA of expected system calls automatically from static source code analysis
  - Runtime monitor compares sequence of issued syscalls to FSA model
  - If an invalid transition is observed, a violation is reported and the program is terminated
- Let's see an example...

System Call Monitoring

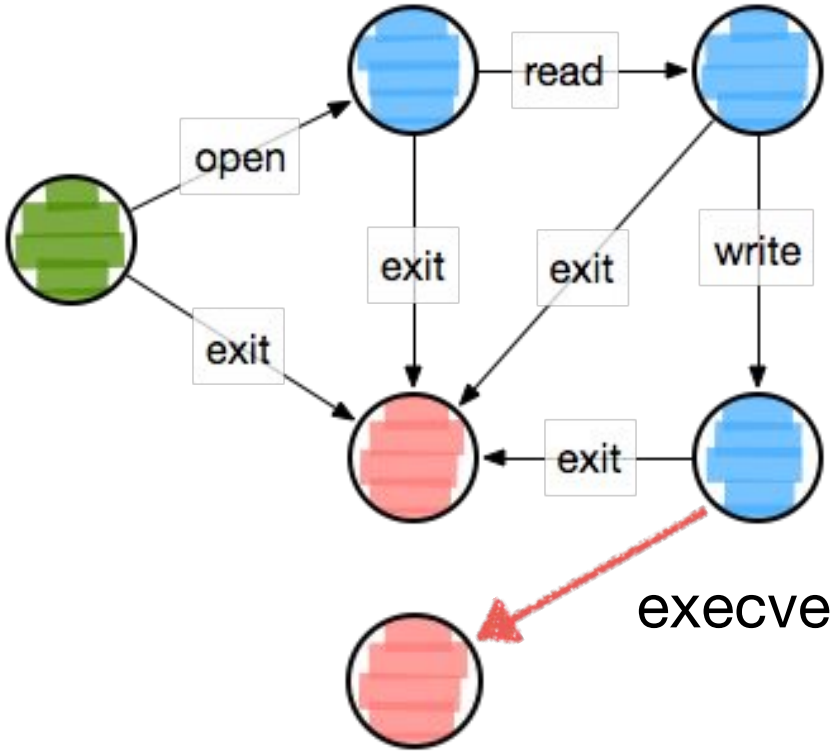
```
int main(int argc, char** argv) {
    if (argc < 2) {
        return 1;
    }

    char buf[4096];
    int fd = open(argv[1], O_RDONLY);
    if (fd < 0) {
        return 1;
    }

    ssize_t n = read(fd, buf, sizeof(buf));
    if (n <= 0) {
        return 1;
    }

    write(1, buf, n);

    return 0;
}
```



Syscall Model Challenges

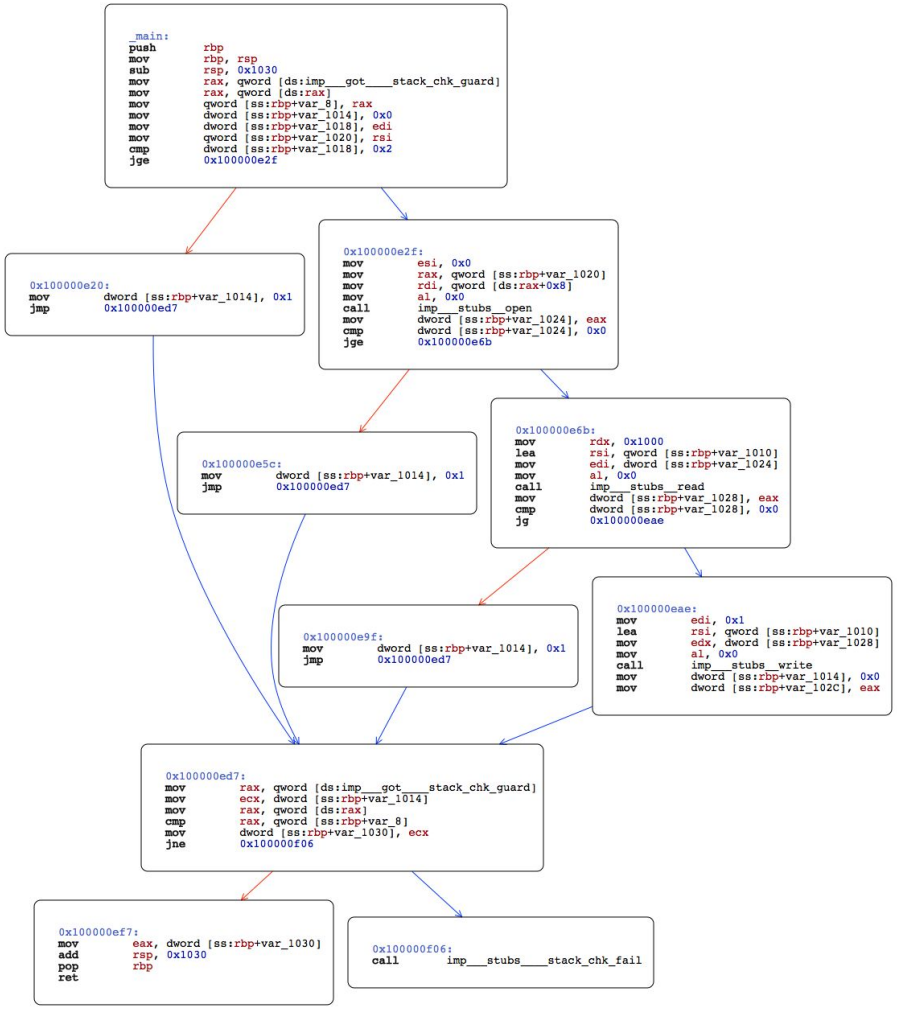
- Original approach required source code
  - But, this isn't always available
  - Later approaches extended this to binary programs, but...
- How precise is the model?
  - If the model is too loose an approximation (i.e., allows too many edges that can't occur in practice) then the attacker has more flexibility to evade the model
  - If the model doesn't contain certain edges, then false positives are possible (with a high cost)
- What if the attacker can construct an attack within the model?

Syscall Model Challenges

```
int main(int argc, char** argv) {
    int (*f)(void) = strtoul(argv[1], NULL, 16);
    return f();
}
```

- What model should be built from the above program?

Binary Syscall Models



- Can build a similar FSA directly from a binary program
  - Construct a control-flow graph (CFG)
  - Map function calls to reachable syscalls
  - Same problems with indirect jumps